Доказательство свойств функциональных программ методом насыщения равенствами

(по диссертации на соискание степени к.ф.-м.н.)

Сергей Александрович Гречаник

ИПМ им. М.В. Келдыша

Научный руководитель:
к.ф.-м.н. Сергей Анатольевич Романенко

Область исследования

Доказательство эквивалентности выражений для функционального языка

Неформально:

- Дана программа P (набор определений)

$$f_1(\overline{x}) = E_1$$
$$\dots$$
$$f_n(\overline{y}) = E_n$$

- Дана формула $E_l \cong E_R$
- Определить, верно ли, что $P \Rightarrow E_l \cong E_R$

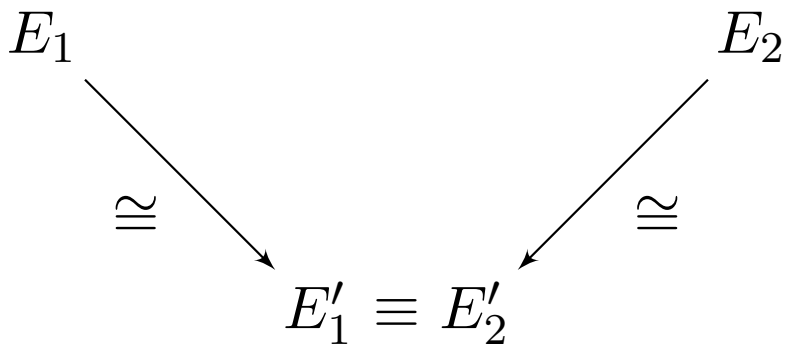
Актуальность

Доказательство эквивалентностей актуально в следующих областях:

- Верификация программ: сами эквивалентности могут быть верифицируемыми свойствами, а также могут использоваться для доказательства других свойств
- Оптимизация программ: эквивалентности могут использоваться для переписывания программ (пример: правила переписывания в GHC)

Трансформационный подход

Преобразуем оба выражения эквивалентными преобразованиями и сравним результаты на синтаксическое совпадение



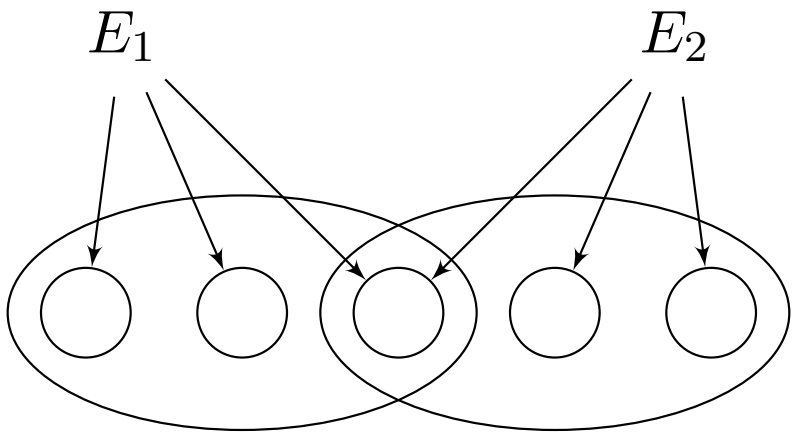
Пример — суперкомпиляция:

$$\mathrm{SC}(E_1) \equiv \mathrm{SC}(E_1) \Rightarrow E_1 \cong E_2$$

- Lisitsa and Webster. “Supercompilation for Equivalence Testing in Metamorphic Computer Viruses Detection”. 2008
- Ключников. “Выявление и доказательство свойств функциональных программ методами суперкомпиляции”. 2010

Многорезультатный подход

Преобразуем оба выражения разными способами, получив два множества, и проверим эти множества на пересечение



Пример — многорезультатная суперкомпиляция:

$$\mathrm{MSC}(E_1) \cap \mathrm{MSC}(E_1) \neq \varnothing \Rightarrow E_1 \cong E_2$$

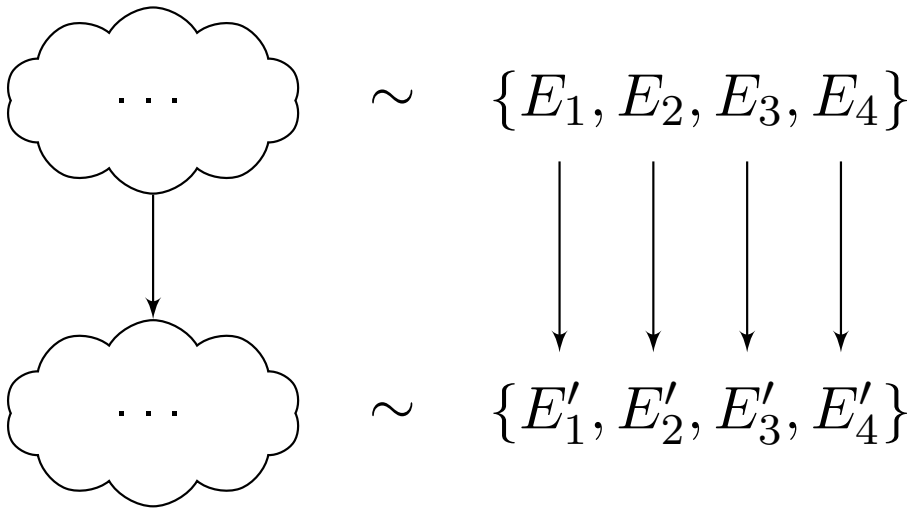
- Klyuchnikov and Romanenko. “Multi-Result Supercompilation as Branching Growth of the Penultimate Level in Metasystem Transitions”. 2012

Проблема: **комбинаторный взрыв** количества результатов

Компактное представление

Вместо множества выражений можно использовать его **компактное представление**

- Занимает меньше места, чем множество
- Шаг преобразования компактного представления соответствует нескольким шагам преобразования множества



Пример — **насыщение равенствами**

- Tate et al. “Equality saturation: a new approach to optimization”. 2009, где описывается система Peggy, реализующая насыщение равенствами для императивных программ

Постановка задачи

- Расширить метод насыщения равенствами, в том числе понятиями и операциями из суперкомпиляции, чтобы он стал применим к нестрогому функциональному языку первого порядка
- Исследовать применимость метода к задаче индуктивного доказательства эквивалентности функций
- Реализовать метод в экспериментальной системе для доказательства эквивалентности и исследовать поведение метода на тестовых примерах

Пример

Дана программа (как система уравнений):

$$(1)\, f(x) = \mathbf{case}\ x\ \mathbf{of}\{Z \to Z; S(x') \to g(x')\}$$
$$(2)\, g(x) = f(f(x))$$

Доказать, что $f(x) = g(x)$
Раскроем внешнюю f в (2) по определению (1):

$$(3)\, g(x) = \mathbf{case}\ f(x)\ \mathbf{of}\{Z \to Z; S(x') \to g(x')\}$$

Раскроем f в (3) и протолкнём внешнее сопоставление с образцом внутрь:

$$\begin{aligned}(4)\, g(x) &= \mathbf{case}\ (\mathbf{case}\ x\ \mathbf{of}\{\ldots\})\ \mathbf{of}\{Z \to Z; S(x') \to g(x')\} = \\ &= \mathbf{case}\ x\ \mathbf{of}\{Z \to \mathbf{case}\ Z\ \mathbf{of}\{\ldots\}; \\ &\qquad S(x') \to \mathbf{case}\ g(x')\ \mathbf{of}\{\ldots\}\} = \\ &= \mathbf{case}\ x\ \mathbf{of}\{Z \to Z; S(x') \to h(x')\} \\ (5)\, h(x) &= \mathbf{case}\ g(x)\ \mathbf{of}\{Z \to Z; S(x') \to g(x')\}\end{aligned}$$

Пример (продолжение)

Получаем следующую систему-следствие:

$$(1)\, f(x) = \mathbf{case}\ x\ \mathbf{of}\{Z \to Z; S(x') \to g(x')\}$$
$$(2)\, g(x) = f(f(x))$$
$$(3)\, g(x) = \mathbf{case}\ f(x)\ \mathbf{of}\{Z \to Z; S(x') \to g(x')\}$$
$$(4)\, g(x) = \mathbf{case}\ x\ \mathbf{of}\{Z \to Z; S(x') \to h(x')\}$$
$$(5)\, h(x) = \mathbf{case}\ g(x)\ \mathbf{of}\{Z \to Z; S(x') \to g(x')\}$$

Выделим из неё две подсистемы:

$$R(f,g) = \begin{cases} (1)\, f(x) = \mathbf{case}\ x\ \mathbf{of}\{Z \to Z; S(x') \to g(x')\} \\ (3)\, g(x) = \mathbf{case}\ f(x)\ \mathbf{of}\{Z \to Z; S(x') \to g(x')\} \end{cases}$$

$$R(g,h) = \begin{cases} (4)\, g(x) = \mathbf{case}\ x\ \mathbf{of}\{Z \to Z; S(x') \to h(x')\} \\ (5)\, h(x) = \mathbf{case}\ g(x)\ \mathbf{of}\{Z \to Z; S(x') \to g(x')\} \end{cases}$$

Система $R(\_, \_)$ в данном случае такова, что имеет не более одного решения, а значит $R(f,g) \wedge R(g,h) \Rightarrow f = g \wedge g = h$:

$$(6)\, f(x) = g(x)$$
$$(7)\, g(x) = h(x)$$

Пример (продолжение)

Теперь исходную систему можно упростить, подставив $g = h = f$ и удалив дубликаты:

$$(1')\, f(x) = \mathbf{case}\ x\ \mathbf{of}\{Z \to Z; S(x') \to f(x')\}$$
$$(2')\, f(x) = f(f(x))$$
$$(3')\, f(x) = \mathbf{case}\ f(x)\ \mathbf{of}\{Z \to Z; S(x') \to f(x')\}$$
$$(6')\, g(x) = f(x)$$

Теперь рассмотрим программу, состоящую из (1'):

$$(1')\, f(x) = \mathbf{case}\ x\ \mathbf{of}\{Z \to Z; S(x') \to f(x')\}$$

Она имеет такой вид, что может быть использована как остаточная программа (для f), эквивалентная исходной, при этом она более эффективна

Входной язык

Входной язык — нестрогий нетипизированный функциональный язык первого порядка.
Программа — набор определений вида $f(x_1, ..., x_n) = E$,
каждая функция имеет ровно одно определение, выражения имеют вид:

$$
\begin{aligned}
E ::= {} & x \\
& \mid f(E_1, ..., E_n) \\
& \mid C(E_1, ..., E_n) \\
& \mid \textbf{case } E_0 \textbf{ of } \{ \overline{C_i(\overline{x_{ij}}) \ \rightarrow \ E_i;} \}
\end{aligned}
$$

Пример программы:

$$
\begin{array}{ll}
\mathrm{add}(x, y) = & \mathrm{mul}(x, y) = \\
\quad \textbf{case } x \textbf{ of} & \quad \textbf{case } x \textbf{ of} \\
\quad\quad Z \rightarrow y & \quad\quad Z \rightarrow Z \\
\quad\quad S(x') \rightarrow S(\mathrm{add}(x', y)) & \quad\quad S(x') \rightarrow \mathrm{add}(y, \mathrm{mul}(x', y))
\end{array}
$$

Семантика входного языка

Введём денотационную семантику в логическом стиле (это позволит использовать её без изменений для компактного представления)

Множество значений — **наибольшая** неподвижная точка:

$$\mathbb{A} = \{C(\overline{a_i}) \mid a_i \in \mathbb{A},\ C \text{ — конструктор}\} \cup \{\bot\}$$

Интерпретация программы — функция $\nu : F(P) \to [\mathbb{A}^n \to \mathbb{A}]$, задающая значения для функций программы из $F(P)$

Значение выражения E в данной программе P, интерпретации программы ν и контексте $\theta = \{\overline{x_i \to a_i}\}$ определяется так:

$$[\![E]\!]_{P,\nu,\theta} : \mathbb{A}$$

$$[\![x]\!]_{P,\nu,\theta} = \theta(x)$$

$$[\![f(\overline{E_i})]\!]_{P,\nu,\theta} = \nu(f)(\overline{[\![E_i]\!]_{P,\nu\theta}})$$

$$[\![C(\overline{E_i})]\!]_{P,\nu,\theta} = C(\overline{[\![E_i]\!]_{P,\nu,\theta}})$$

$$[\![\mathbf{case}\ \ldots\ \mathbf{of}\ \{\ldots\}]\!]_{P,\nu,\theta} = \ldots \text{ (опущено)}$$

Семантика входного языка: модель

Моделью программы P назовём такую интерпретацию μ, что для всех определений $f(\overline{x_i}) = E$ из P выполнено $\forall\theta[\![f(\overline{x_i})]\!]_{P,\mu,\theta} = [\![E]\!]_{P,\mu,\theta}$

Стандартный смысл программы определяется её *наименьшей* моделью (относительно $\sqsubseteq$)

Обозначим $E_1 \cong_{P,\nu} E_2 \overset{\text{def}}{\Leftrightarrow} \forall\theta[\![E_1]\!]_{P,\nu,\theta} = [\![E_2]\!]_{P,\nu,\theta}$

Компактное представление множества функциональных программ

Компактное представление отличается от функциональной программы только двумя вещами:

- Разрешено множественное определение функций
- Каждое определение имеет простейший вид $f(\overline{x_i}) = S$, где

$$
\begin{aligned}
S ::= & \; R \\
& \mid x \\
& \mid f(\overline{R_i}) \\
& \mid C(\overline{R_i}) \\
& \mid \textbf{case } R_0 \textbf{ of } \{ \; \overline{C_i(\overline{x_{ij}}) \; \rightarrow \; R_i;} \; \} \\
R ::= & \; f(\overline{x_i}) \text{ , где } x_i \text{ — } \textbf{различные} \text{ переменные}
\end{aligned}
$$

Второе требование нужно для повышения шеринга общих подвыражений между определениями и легко удовлетворяется введением промежуточных функций

Компактное представление: пример

Идея в том, что множественные определения должны совпадать по смыслу (но могут отличаться, например, по производительности)

$$\mathrm{id}(x) = x$$
$$\mathrm{true}() = \mathrm{True}$$
$$\mathrm{false}() = \mathrm{False}$$
$$\mathrm{not}(t) = \textbf{case}\ id(t)\ \textbf{of}$$
$$\quad \mathrm{True} \to \mathrm{false}()$$
$$\quad \mathrm{False} \to \mathrm{true}()$$

$$\mathrm{even}(x) = \textbf{case}\ id(x)\ \textbf{of}$$
$$\quad Z \to \mathrm{true}()$$
$$\quad S(x') \to \mathrm{odd}(x')$$
$$\mathrm{odd}(x) = \textbf{case}\ id(x)\ \textbf{of}$$
$$\quad Z \to \mathrm{false}()$$
$$\quad S(x') \to \mathrm{even}(x')$$
$$\mathrm{odd}(x) = \mathrm{not}(\mathrm{even}(x))$$

Выбирая по одному определению для каждой функции, можно выделить программу из компактного представления

Компактное представление как множество программ

Иногда из на первый взгляд нормального компактного представления можно выделить **неэквивалентные** программы (с разными наименьшими моделями)

$$
\begin{aligned}
&\mathrm{even}(x) = \textbf{case } id(x) \textbf{ of}\\
&\quad Z \to \mathrm{true}()\\
&\quad S(x') \to \mathrm{odd}(x')\\
&\mathrm{even}(x) = \mathrm{not}(\mathrm{odd}(x))\\
&\mathrm{odd}(x) = \textbf{case } id(x) \textbf{ of}\\
&\quad Z \to \mathrm{false}()\\
&\quad S(x') \to \mathrm{even}(x')\\
&\mathrm{odd}(x) = \mathrm{not}(\mathrm{even}(x))
\end{aligned}
$$

$$
\begin{aligned}
&\mathrm{even}(x) = \textbf{case } id(x) \textbf{ of}\\
&\quad Z \to \mathrm{true}()\\
&\quad S(x') \to \mathrm{odd}(x')\\
&\mathrm{odd}(x) = \mathrm{not}(\mathrm{even}(x))
\end{aligned}
$$

$$\neq$$

$$
\begin{aligned}
&\mathrm{even}(x) = \mathrm{not}(\mathrm{odd}(x))\\
&\mathrm{odd}(x) = \mathrm{not}(\mathrm{even}(x))
\end{aligned}
$$

Поэтому при выделении программы надо использовать дополнительные ограничения (см. далее)

Эквивалентные преобразования компактного представления

Пусть G_1 и G_2 — компактные представления, F — множество имён функций, тогда $G_1 \cong_\cap G_2$, если для любой модели μ_1 представления G_1 существует модель μ_2 представления G_2 такая, что $\mu_1(f) = \mu_2(f), f \in F(G_1) \cap F(G_2)$, и наоборот.

Можно заменять части компактных представлений на эквивалентные: если $G_1 \cong_\cap G_2$ и $G_1 \subseteq G$, то $G \cong_\cap (G \setminus G_1) \cup G_2'$, где G_2' получено из G_2 заменой всех имён функций не из G_1 на свежие (не из G).

Если бы вместо всех моделей мы бы рассматривали только наименьшие, последнее свойство не выполнялось бы

Доказательство эквивалентности насыщением равенствами

Дано:

- Программа P
- Пара выражений E_1 и E_2

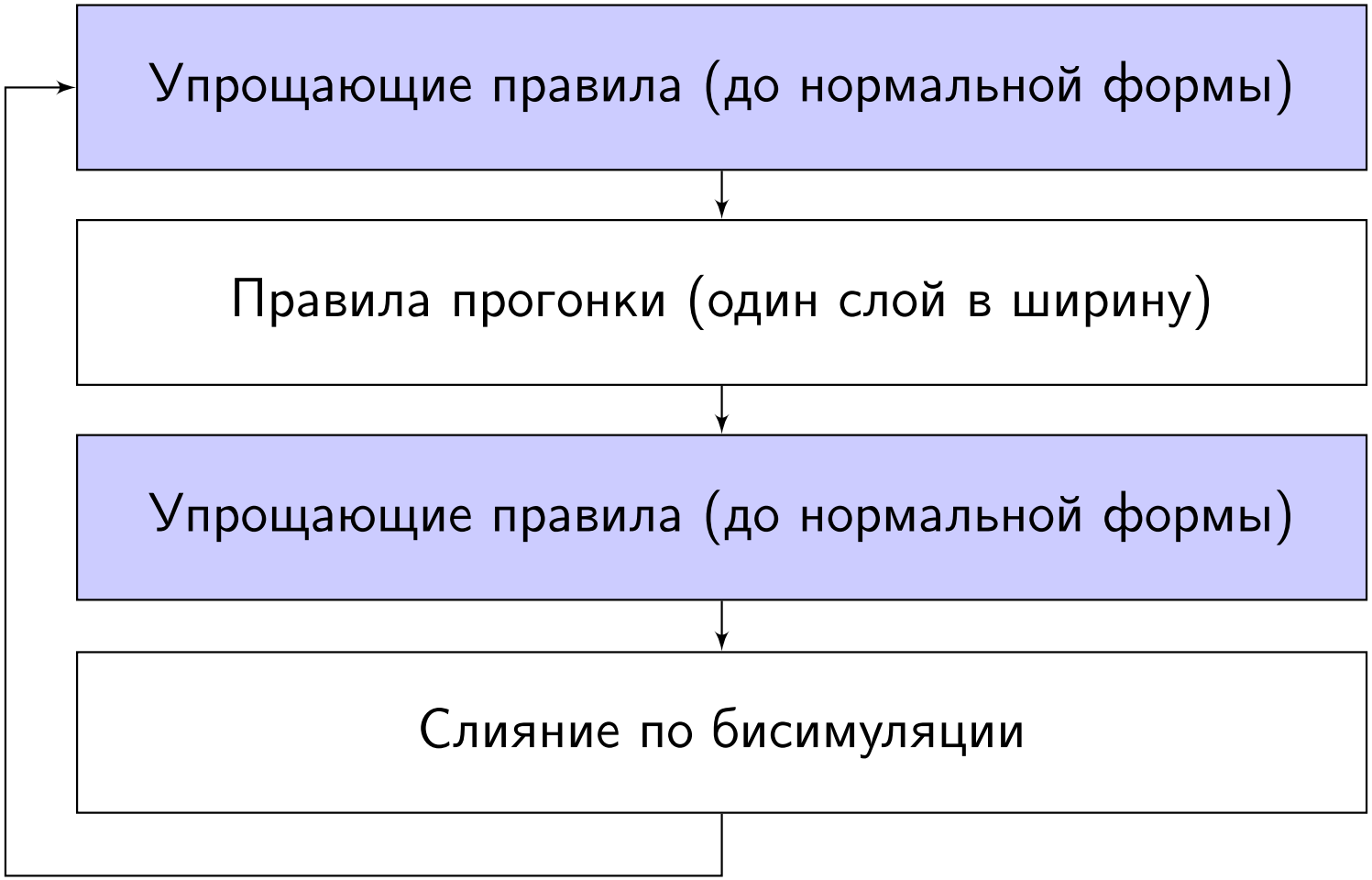
Доказать, что $E_1 \cong_{P,\hat{\mu}} E_2$, где $\hat{\mu}$ — наименьшая модель

- Преобразуем P в компактное представление G введением промежуточных функций, а также добавим два определения: $f_1(\overline{\mathrm{FV}}(E_1)) = E_1$ и $f_2(\overline{\mathrm{FV}}(E_2)) = E_2$
- Насыщение: преобразуем G в G' эквивалентными преобразованиями ($G \cong_{\cap} G'$) до тех пор, пока в G' не будет содержаться определение $f_1(\overline{x}) = f_2(\overline{x})$.

Наличие такого определения будет означать, что $\mu(f_1) = \mu(f_2)$ для всех моделей G, а значит и для всех моделей P, а значит и для $\hat{\mu}$.

Порядок применения преобразований

Преобразования разбиты на несколько групп



Упрощающие правила (1)

Упрощающие правила уменьшают граф, они могут удалять определения. Их завершаемость (с точностью до дублирования определений) и конфлюэнтность (с некоторыми ограничениями) доказываются в диссертации.

Все правила применяются с точностью до переименования функций и переменных

Удаление дубликатов

$$\left\{ \begin{array}{l} f(\overline{x}) = E \\ f(\overline{x}) = E \end{array} \right\} \mapsto \left\{ \begin{array}{l} f(\overline{x}) = E \end{array} \right\}$$

Упрощающие правила (2)

Переклейка

$$\left\{\begin{array}{l} f(\overline{x_i}) = g(\overline{x_j}) \\ D\langle f(\overline{e_i})\rangle \end{array}\right\} \mapsto \left\{\begin{array}{l} f(\overline{x_i}) = g(\overline{x_j}) \\ D\langle g(\overline{e_j})\rangle \end{array}\right\}$$

($f(\overline{e_i})$ в D может быть как слева так и справа от $=$)

На практике применяется полная переклейка — последовательность всевозможных переклеек вдоль одного и того же равенства, рассматриваемая как один шаг преобразования (это влияет на завершаемость). Полную переклейку с последующим удалением дубликатов назовём *слиянием*.

Транзитивность

$$\left\{\begin{array}{l} f(\overline{x_i}) = E \\ g(\overline{x_j}) = E \end{array}\right\} \mapsto \left\{\begin{array}{l} f(\overline{x_i}) = E \\ g(\overline{x_j}) = E \\ f(\overline{x_i}) = g(\overline{x_j}) \end{array}\right\}$$

В сочетании с переклейкой даёт конгруэнтное замыкание

Упрощающие правила (3)

Нормализация определения

$$\{\ f(\overline{x_i}) = E\langle\overline{x_j}\rangle\ \} \mapsto \{\ f(\overline{x_k}) = E\langle x_1, ..., x_n\rangle\ \}$$

Переименовывает переменные в определении так, чтобы они шли слева направо в теле

Позволяет сливать по конгруэнтности определения, которые иначе не были бы слиты

Пример слияния по конгруэнтности

$$\left\{\begin{array}{l} f_1(x,y) = C(h_1(x,y), h_2(x,y)) \\ f_2(x,y) = C(h_2(x,y), h_1(x,y)) \\ h_1(x,y) = D(x,y) \\ h_2(x,y) = D(y,x) \end{array}\right\} \mapsto \left\{\begin{array}{l} f_1(x,y) = C(h_1(x,y), h_2(x,y)) \\ f_2(x,y) = C(h_2(x,y), h_1(x,y)) \\ h_1(x,y) = D(x,y) \\ h_2(y,x) = D(x,y) \end{array}\right\} \mapsto$$

$$\left\{\begin{array}{l} f_1(x,y) = C(h_1(x,y), h_1(y,x)) \\ f_2(x,y) = C(h_1(y,x), h_1(x,y)) \\ h_1(x,y) = D(x,y) \\ h_1(x,y) = D(x,y) \\ h_2(y,x) = h_1(x,y) \end{array}\right\} \mapsto \left\{\begin{array}{l} f_1(x,y) = C(h_1(x,y), h_1(y,x)) \\ f_2(y,x) = C(h_1(x,y), h_1(y,x)) \\ h_1(x,y) = D(x,y) \\ h_2(y,x) = h_1(x,y) \end{array}\right\} \mapsto$$

$$\left\{\begin{array}{l} f_1(x,y) = C(h_1(x,y), h_1(y,x)) \\ f_2(y,x) = C(h_1(x,y), h_1(y,x)) \\ h_1(x,y) = D(x,y) \\ h_2(y,x) = h_1(x,y) \end{array}\right\} \mapsto \left\{\begin{array}{l} f_1(x,y) = C(h_1(x,y), h_1(y,x)) \\ h_1(x,y) = D(x,y) \\ h_2(y,x) = h_1(x,y) \\ f_2(y,x) = f_1(x,y) \end{array}\right\}$$

Упрощающие правила (4)

Преобразование вызова в перестановку параметров

$$\left\{\begin{array}{l} f(\overline{x_i}) = g(\overline{id(x_j)}) \\ id(x) = x \end{array}\right\} \mapsto \left\{\begin{array}{l} f(\overline{x_i}) = g(\overline{x_j}) \\ id(x) = x \end{array}\right\}$$

Выполняется только если все x_j различны

Редукция вызова тождественной функции

$$\left\{\begin{array}{l} f(\overline{x_i}) = id(g(\overline{x_j})) \\ id(x) = x \end{array}\right\} \mapsto \left\{\begin{array}{l} f(\overline{x_i}) = g(\overline{x_j}) \\ id(x) = x \end{array}\right\}$$

Смысл: $(\lambda x.x)(g(x)) = g(x)$

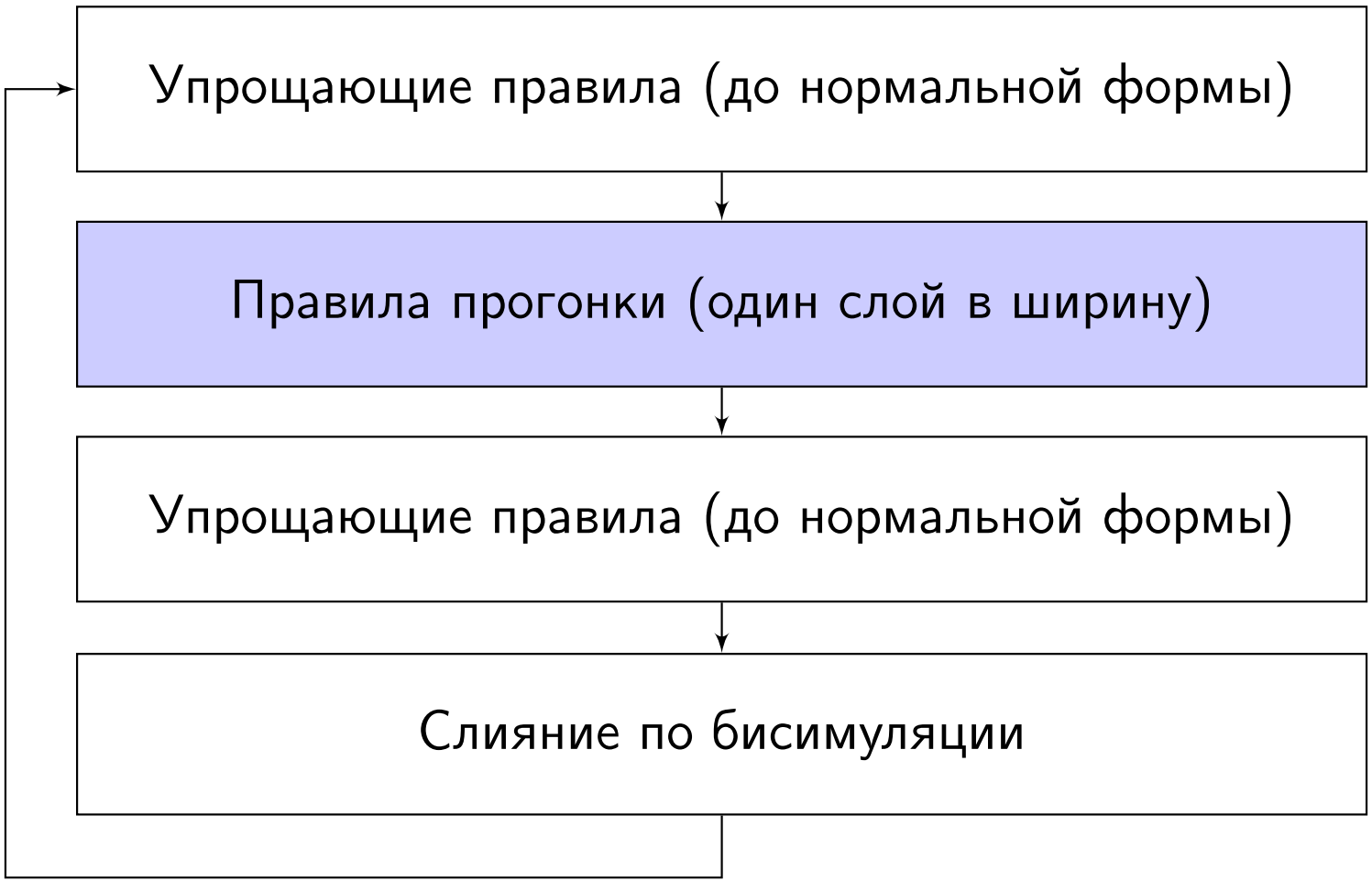
Упрощающие правила (5)

Редукция сопоставления с образцом

$$\left\{\begin{array}{l} f(\overline{x_i}) = \mathbf{case}\ c(\overline{x_j})\ \mathbf{of}\{C(\overline{y_k}) \to h(\overline{y_k}, \overline{x_m}); \ldots\} \\ c(\overline{x_j}) = C(\overline{g_k(\overline{x_l})}) \\ id(x) = x \end{array}\right\} \mapsto$$

$$\left\{\begin{array}{l} f(\overline{x_i}) = h(\overline{g_k(\overline{x_l})}, \overline{id(x_m)}) \\ c(\overline{x_j}) = C(\overline{g_k(\overline{x_l})}) \\ id(x) = x \end{array}\right\}$$

Смысл: $\mathbf{case}\ C(g(x))\ \mathbf{of}\{C(y) \to h(y, x)\} = h(g(x), x)$


Упрощающие правила (до нормальной формы)
Правила прогонки (один слой в ширину)
Упрощающие правила (до нормальной формы)
Слияние по бисимуляции

Правила прогонки

Правила прогонки никогда не удаляют определения, и применяются послойно в ширину (применения правил в одном слое не зависят друг от друга), поэтому от них не требуется завершаемость, а конфлюэнтность выполняется автоматически (как в классическом насыщении равенствами)

Прогонка: редукция вызова функции

Вызов функции работает как явная подстановка

Редукция вызова функции, состоящей из вызова функции

$$\left\{\begin{array}{l} f(\overline{x_i}) = g(\overline{h_j(\overline{x_l})}) \\ g(\overline{y_j}) = s(\overline{t_k(\overline{y_m})}) \end{array}\right\} \mapsto \left\{\begin{array}{l} \overline{f(\overline{x_i}) = s(\overline{r_k(\overline{x_i})})} \\ \overline{r_k(\overline{x_i}) = t_k(\overline{h_m(\overline{x_l})})} \\ f(\overline{x_i}) = g(\overline{h_j(\overline{x_l})}) \\ g(\overline{y_j}) = s(\overline{t_k(\overline{y_m})}) \end{array}\right\}$$

Смысл: $(\lambda y.s(t(y)))(h(x)) = s(t(h(x)))$

Редукция вызова функции, состоящей из конструктора

$$\left\{\begin{array}{l} f(\overline{x_i}) = g(\overline{h_j(\overline{x_l})}) \\ g(\overline{y_j}) = C(\overline{t_k(\overline{y_m})}) \end{array}\right\} \mapsto \left\{\begin{array}{l} \overline{f(\overline{x_i}) = C(\overline{r_k(\overline{x_i})})} \\ \overline{r_k(\overline{x_i}) = t_k(\overline{h_m(\overline{x_l})})} \\ f(\overline{x_i}) = g(\overline{h_j(\overline{x_l})}) \\ g(\overline{y_j}) = C(\overline{t_k(\overline{y_m})}) \end{array}\right\}$$

Смысл: $(\lambda y.(t(y)))(h(x)) = (t(h(x)))$

Редукция вызова функции, состоящей из сопоставления с образцом аналогично, опущено для краткости

Прогонка: распространение позитивной информации

Распространение позитивной информации

$$\left\{\begin{array}{l} f(\overline{x_i}) = \textbf{case}\ id(x_n)\ \textbf{of}\{\overline{C_k(\overline{y_l}) \to g_k(\overline{y_l}, \overline{x_j})}\} \\ id(x) = x \end{array}\right\} \mapsto$$

$$\mapsto \left\{\begin{array}{l} f(\overline{x_i}) = \textbf{case}\ id(x_n)\ \textbf{of}\{\overline{C_k \to h_k(\overline{y_l}, \overline{x_i})}\} \\ \overline{h_k(\overline{y_l}, \overline{x_i}) = g_k(\overline{y_l}, \ldots, id(x_{n-1}), c_k(\overline{y_l}), id(x_{n+1}), \ldots)} \\ \overline{c_k(\overline{y_l}) = C_k(\overline{y_l})} \\ f(\overline{x_i}) = \textbf{case}\ id(x_n)\ \textbf{of}\{\overline{C_k \to g_k(\overline{x_j})}\} \\ id(x) = x \end{array}\right\}$$

Смысл:

$\textbf{case}\ x\ \textbf{of}\{C(y) \to h(y, x)\} = \textbf{case}\ x\ \textbf{of}\{C(y) \to h(y, C(y))\}$

Прогонка: ассоциативность сопоставления с образцом

Ассоциативность сопоставления с образцом

$$\left\{\begin{array}{l} f(\overline{x_i}) = \textbf{case}\ g(\overline{x_i})\ \textbf{of}\{\overline{C_k(\overline{y_l}) \to h_k(\overline{y_l}, \overline{x_i})}\} \\ g(\overline{x_i}) = \textbf{case}\ s(\overline{x_i})\ \textbf{of}\{\overline{D_n(\overline{z_m}) \to t_n(\overline{z_m}, \overline{x_i})}\} \\ id(x) = x \end{array}\right\} \mapsto$$

$$\mapsto \left\{\begin{array}{l} f(\overline{x_i}) = \textbf{case}\ s(\overline{x_i})\ \textbf{of}\{\overline{D_n(\overline{z_m}) \to r_n(\overline{z_m}, \overline{x_i})}\} \\ \hline r_n(\overline{z_m}, \overline{x_i}) = \textbf{case}\ t_n(\overline{z_m}, \overline{x_i})\ \textbf{of}\{\overline{C_k(\overline{y_l}) \to h_k(\overline{y_l}, \overline{x_i})}\} \\ f(\overline{x_i}) = \textbf{case}\ g(\overline{x_i})\ \textbf{of}\{\overline{C_k(\overline{y_l}) \to h_k(\overline{y_l}, \overline{x_i})}\} \\ g(\overline{x_i}) = \textbf{case}\ s(\overline{x_i})\ \textbf{of}\{\overline{D_n(\overline{z_m}) \to t_n(\overline{z_m}, \overline{x_i})}\} \\ id(x) = x \end{array}\right\}$$

Смысл:

$\textbf{case}\ (\textbf{case}\ s(x)\ \textbf{of}\{C(y) \to h(y,x)\})\ \textbf{of}\{D(z) \to t(z,x)\} =$
$\textbf{case}\ s(x)\ \textbf{of}\{C(y) \to \textbf{case}\ h(y,x)\ \textbf{of}\{D(z) \to t(z,x)\}\}$

Пример (1)

Рассмотрим пример с ассоциативностью сложения, упрощающие шаги преобразования будем опускать

$$(1)\ \mathrm{add}(x, y) = \mathbf{case}\ id(x)\ \mathbf{of}\{Z \to id(y); S(x') \to \mathrm{sadd}(x', y)\}$$
$$(2)\ \mathrm{sadd}(x, y) = S(\mathrm{add}(x, y))$$
$$(3)\ f(x, y, z) = \mathrm{add}(\mathrm{add}(x, y), id(z))$$

Применим редукцию вызова к (3) и (1):

$$(4)\ f(x, y, z) = \mathbf{case}\ \mathrm{add}(x, y)\ \mathbf{of}\{Z \to id(z); S(x') \to \mathrm{sadd}(x', z)\}$$

Применим ассоциативность сопоставления с образцом к (4) и (1):

$$(5)\ f(x, y, z) = \mathbf{case}\ id(x)\ \mathbf{of}\{Z \to f_1(y, z); S(x') \to f_2(x', y, z)\}$$
$$(6)\ f_1(y, z) = \mathbf{case}\ id(y)\ \mathbf{of}\{Z \to id(z); S(y') \to \mathrm{sadd}(y', z)\}$$
$$(7)\ f_2(x', y, z) = \mathrm{sadd}(\mathrm{add}(x', y), id(z))$$

(6) совпадает с (1) с точностью до имён переменных, поэтому заменяем f_1 на add:

$$(5)\ f(x, y, z) = \mathbf{case}\ id(x)\ \mathbf{of}\{Z \to \mathrm{add}(y, z); S(x') \to f_2(x', y, z)\}$$

Пример (2)

Состояние компактного представления:

$$(1)\ \mathrm{add}(x, y) = \mathbf{case}\ id(x)\ \mathbf{of}\{Z \to id(y); S(x') \to \mathrm{sadd}(x', y)\}$$
$$(2)\ \mathrm{sadd}(x, y) = S(\mathrm{add}(x, y))$$
$$(3)\ f(x, y, z) = \mathrm{add}(\mathrm{add}(x, y), id(z))$$
$$(4)\ f(x, y, z) = \mathbf{case}\ \mathrm{add}(x, y)\ \mathbf{of}\{Z \to id(z); S(x') \to \mathrm{sadd}(x', z)\}$$
$$(5)\ f(x, y, z) = \mathbf{case}\ id(x)\ \mathbf{of}\{Z \to \mathrm{add}(y, z); S(x') \to f_2(x', y, z)\}$$
$$(7)\ f_2(x', y, z) = \mathrm{sadd}(\mathrm{add}(x', y), id(z))$$

Перейдём к f_2. Применим редукцию вызова к (7) и (2):

$$(8)\ f_2(x', y, z) = S(f_3(x', y, z))$$
$$(9)\ f_3(x', y, z) = \mathrm{add}(\mathrm{add}(x', y), id(z))$$

(9) — то же, что и (3) с точностью до переименования переменных, поэтому заменяем f_3 на f слиянием:

$$(8)\ f_2(x', y, z) = S(f(x', y, z))$$

Пример (3)

В итоге получаем следующее компактное представление:

(1) $\mathrm{add}(x, y) = \mathbf{case}\ id(x)\ \mathbf{of}\{Z \to id(y); S(x') \to \mathrm{sadd}(x', y)\}$

(2) $\mathrm{sadd}(x, y) = S(\mathrm{add}(x, y))$

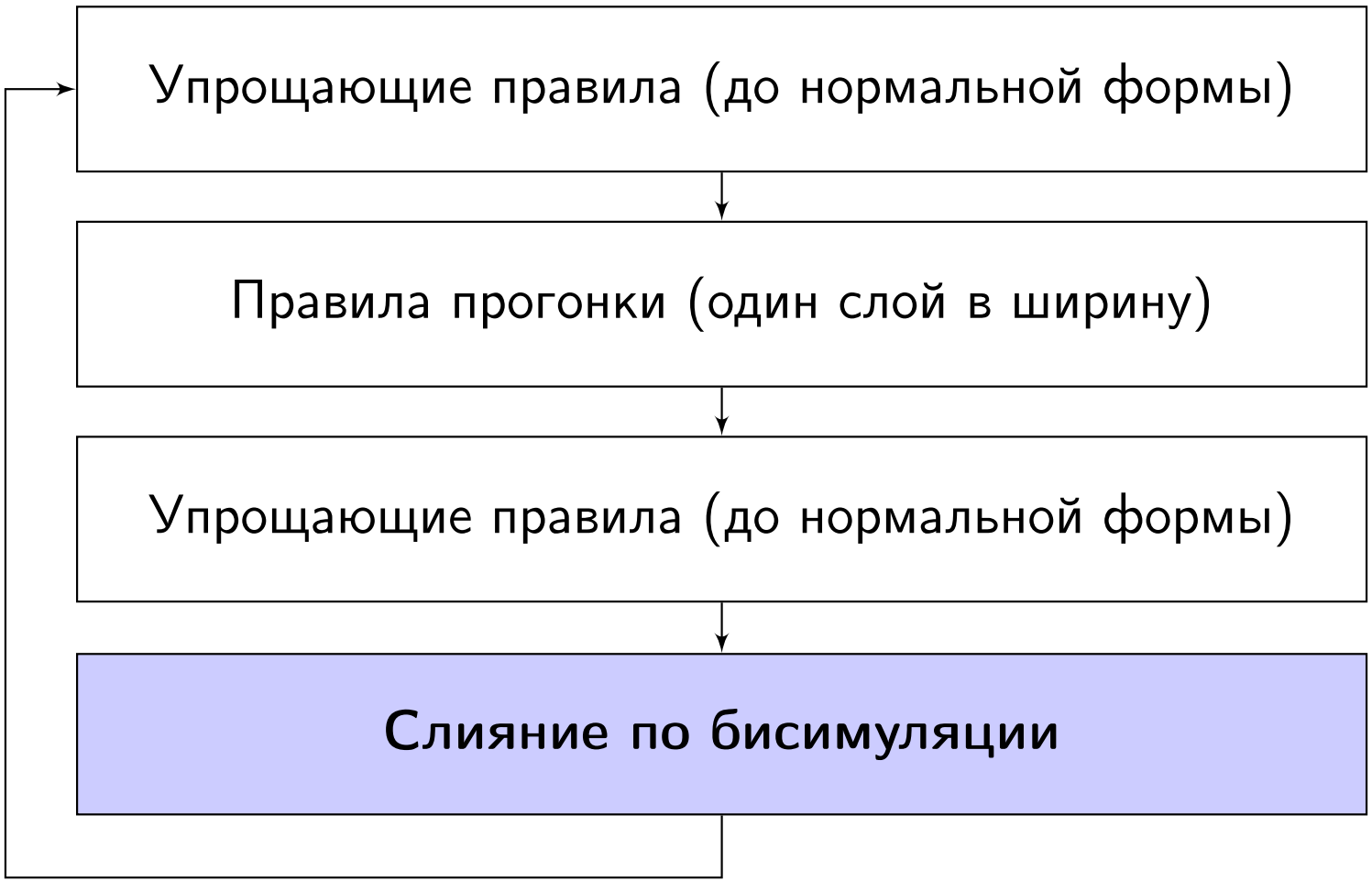
(3) $f(x, y, z) = \mathrm{add}(\mathrm{add}(x, y), id(z))$

(4) $f(x, y, z) = \mathbf{case}\ \mathrm{add}(x, y)\ \mathbf{of}\{Z \to id(z); S(x') \to \mathrm{sadd}(x', z)\}$

(5) $f(x, y, z) = \mathbf{case}\ id(x)\ \mathbf{of}\{Z \to \mathrm{add}(y, z); S(x') \to f_2(x', y, z)\}$

(7) $f_2(x', y, z) = \mathrm{sadd}(\mathrm{add}(x', y), id(z))$

(8) $f_2(x', y, z) = S(f(x', y, z))$


Упрощающие правила (до нормальной формы)
Правила прогонки (один слой в ширину)
Упрощающие правила (до нормальной формы)
Слияние по бисимуляции

Слияние по бисимуляции: идея

Из следующей программы нельзя вывести $f() = g()$ введёнными ранее преобразованиями:

$$f() = S(f())$$
$$g() = S(g())$$

Нужно преобразование, являющееся аналогом резидуализации с последующим синтаксическим сравнением: если из компактного представления можно выделить две *хорошие* программы, совпадающие с точностью до переименования функций, то соответствующие функции равны

Слияние по бисимуляции: корректность

Хорошая программа – та, которая имеет единственную модель
Программы $f() = S(f())$ и $g() = S(g())$ хорошие, поэтому можно вывести равенство $g() = f()$ и произвести переклейку
Программа $f(x) = f(f(x))$ плохая:

$$f(x) = C()$$
$$f(x) = f(f(x))$$
$$g(x) = D()$$
$$g(x) = g(g(x))$$

Проверить, является ли программа хорошей можно при помощи признаков структурной и защищённой рекурсии, как в Agda (Abel and Altenkrich. “A Predicative Analysis of Structural Recursion”. 2002)
В диссертации доказывается, что эти признаки можно применять и для нетотального случая без разделения на данные и коданные

Слияние по бисимуляции: бисимуляция

Точного совпадения двух программ не требуется, достаточно бисимуляции, т.е. отношения R над именами функций такого, что если $f_0\ R\ g_0$ и $(f(\ldots) = E\langle f_1 ... f_n\rangle) \in P_1$, то $(g(\ldots) = E\langle g_1 ... g_n\rangle) \in P_2$ для некоторых $g_1, ..., g_n$ таких, что $f_i\ R\ g_i$, и наоборот.

Пример:

$$P_1 = \{\ f() = C(f(), f())\}$$

$$P_2 = \left\{ \begin{array}{l} h() = C(g(), h()) \\ g() = C(h(), h()) \end{array} \right\}$$

$$R = \{(f, h), (f, g)\}$$

Слияние по бисимуляции: алгоритм

Задача поиска двух подграфов в отношении бисимуляции является NP-полной Используем поиск в глубину с некоторыми эвристиками Алгоритм поиска пары программ в отношении бисимуляции, определяющих две заданные функции

- Вход: имена f_1, f_2 и история посещённых определений
- Если $f_1 \equiv f_2$, то возвращаем $(\{\}, \{\})$
- Если пара (f_1, f_2) есть в истории, то возвращаем $(\{\}, \{\})$
- Для каждой пары определений $(f_1(\overline{x}) = E\langle\overline{g_i}\rangle) \in P$ и $(f_2(\overline{x}) = E\langle\overline{h_i}\rangle) \in P$:
  - Рекурсивно производим поиск для h_i и g_i
  - Если для всех i поиск вернул пары $(P_i,\ P_i')$, то возвращаем $(\{(f_1(\overline{x}) = E)\} \cup \bigcup P_i, \{(f_2(\overline{x}) = E)\} \cup \bigcup P_i')$
  - Если хотя бы для одной пары поиск завершился неуспешно, переходим к следующей паре определений
- Если для всех пар определений поиск завершился неуспешно, возвращаем неуспех

Слияние по бисимуляции: алгоритм

- Если функции имеют несовместимые определения, например $(f_1(\overline{x}) = C_1(...)) \in P$ и $(f_2(\overline{x}) = C_2(...)) \in P$, причём $C_1 \neq C_2$, то возвращаем неуспех
- Отрицательный результат поиска кэшируются (положительный сразу приводит к слиянию)
- Проверку признаков структурной и защищённой рекурсии можно *частично* производить в процессе поиска при свёртке, что позволит отсечь заведомо плохие программы (но не будет гарантировать, что они хорошие)

Слияние по бисимуляции: леммы

На этапе применения слияния по бисимуляции на наличие бисимуляций тестируются *все* пары вершин

Это даёт эффект автоматического порождения и доказательства вспомогательных лемм, как в многоуровневой суперкомпиляции

- Klyuchnikov and Romanenko. "Towards Higher-Level Supercompilation". 2010
- Hamilton. "Distillation: extracting the essence of programs". 2007

Эффект ограничен из-за отсутствия сложных обобщений, в основном наблюдается на искусственных примерах (вроде $f(x) = f(f(x))$, рассмотренного в начале презентации)

Экспериментальные результаты

Количество взятых примеров на наборе тестов (наш набор + некоторые примеры из TIP):

Наш прувер	56
HOSC	51
Zeno	66
HipSpec	73
Всего	99

Сильные стороны:

- Произвольная глубина снятия конструкторов при доказательстве по индукции
- Поддержка коиндукции из коробки

Слабые стороны:

- Нет сложных обобщений (только откусывание внешнего вызова функции)
- Нет поддержки импликаций

Примеры, которые берутся прувером

$x + (y + z) = (x + y) + z$

$x * (y * z) = (x * y) * z$

$(x + z) * y = x * y + z * y$

$\text{length}(\text{concat}(x)) = \text{sum}(\text{map}(\text{length}(x))$

$\text{even}(x) = \text{evenSlow}(x)$ (через not)

$\text{or}(\text{even}(x), \text{odd}(x)) = \text{True}$

$f(x) = f(f(x))$ (из начала презентации)

$\text{cycle}([A, B]) = \text{Cons}(A, \text{cycle}([B, A]))$

Примеры, которые НЕ берутся прувером

Требуют больше драйвинга в глубину:

$$\mathrm{unchurch}(\mathrm{churchAdd}(\mathrm{church}(x), \mathrm{church}(y))) = x + y$$

КМП-тест

Требует импликаций:

$$\mathrm{filter}(p, \mathrm{filter}(p, xs)) = \mathrm{filter}(p, xs)$$

Требуют обобщений:

$$\mathrm{even}(\mathrm{doubleAcc}(x, S(y))) = \mathrm{odd}(\mathrm{doubleAcc}(x, y))$$

$$\mathrm{naiveReverse}(xs) = \mathrm{reverse}(xs)$$

$$\mathrm{map}(f, \mathrm{iterate}(f, a)) = \mathrm{iterate}(f, f(a))$$

По техническим причинам:

$$\mathrm{length}(\mathrm{intersperse}(x, xs)) = \mathrm{length}(\mathrm{intersperse}(y, xs))$$

Основные результаты

- На основе насыщения равенствами и суперкомпиляции был разработан метод преобразования программ на нестрогом функциональном языке первого порядка
  - На основе прогонки из суперкомпиляции были разработаны правила преобразования
  - Насыщение равенствами было расширено возможностью доказывать эквивалентности по индукции при помощи слияния по бисимуляции
  - Была экспериментально показана необходимость деструктивно применять некоторые правила
  - Был разработан метод применения правил с точностью до перестановки параметров функций, позволяющий устранять дублирование подпрограмм, отличающихся лишь порядком параметров функций
- На основе разработанного метода была реализована и протестирована система доказательства эквивалентности

Публикации

- Sergei A. Grechanik. “Overgraph Representation for Multi-Result Supercompilation”. In: *Proceedings of the Third International Valentin Turchin Workshop on Metacomputation*. Ed. by A.V. Klimov and S.A. Romanenko. Pereslavl-Zalessky, Russia: Pereslavl-Zalessky: Ailamazyan University of Pereslavl, July 2012, pp. 48–65
- Sergei A. Grechanik. “Inductive Prover Based on Equality Saturation for a Lazy Functional Language”. In: *Ershov Memorial Conference*. Ed. by Andrei Voronkov and Irina Virbitskaite. Vol. 8974. Lecture Notes in Computer Science. Springer, 2014, pp. 127–141
- Sergei Grechanik. “Inductive Prover Based on Equality Saturation”. In: *Proceedings of the Fourth International Valentin Turchin Workshop on Metacomputation*. Ed. by A.V. Klimov and S.A. Romanenko. Pereslavl-Zalessky, Russia: Pereslavl Zalessky: Publishing House "University of Pereslavl", July 2014
- (ВАК) С. А. Гречаник. “Доказательство свойств функциональных программ методом насыщения равенствами”. в: *Программирование* 3 (2015), с. 44—61
- Sergei A. Grechanik. “Proving properties of functional programs by equality saturation”. In: *Programming and Computer Software* 41.3 (2015), pp. 149–161

Далее идут бонусные слайды

* Отличие рассмотрения всех моделей от наименьшей

Некоторые программы зацикливаются хорошо, а некоторые плохо

Имеет бесконечно много моделей, наименьшая из которых $\mu(f) = \bot$:

$$\mathrm{f}() = \mathrm{f}()$$

Имеет ровно одну модель $\mu(f) = \bot$:

$$\begin{aligned}
&\mathrm{inf}() = S(\mathrm{inf}()) \\
&\mathrm{eat}(x) = \mathbf{case}\ x\ \mathbf{of} \\
&\qquad S(x') \to \mathrm{eat}(x') \\
&\mathrm{f}() = \mathrm{eat}(\mathrm{inf}())
\end{aligned}$$

Описанный метод может работать с программами второго типа, но для программ первого типа он слишком слаб, т.к. рассматривает все модели, а не только наименьшую

* Что делать?

Способ 1

Превратить программы, которые плохо зацикливаются, в программы, которые хорошо зацикливаются

$$\left\{\begin{array}{l} f_1(x) = E_1 \\ \ldots \\ f_n(x) = E_n \end{array}\right\} \mapsto$$

$$\mapsto \left\{\begin{array}{l} f_1(x) = \mathrm{Tick}(E_1) \\ \ldots \\ f_n(x) = \mathrm{Tick}(E_n) \\ \mathrm{eatTicks}(x) = \mathbf{case}\ x\ \mathbf{of} \\ \qquad \dfrac{Tick(x') \to \mathrm{eatTicks}(x')}{C_k(\overline{y}) \to C_k(\overline{\mathrm{eatTicks}(y)})} \end{array}\right\}$$

А задание $E_1 \cong E_2$ переходит в $\mathrm{eatTicks}(E_1) \cong \mathrm{eatTicks}(E_2)$

* Что делать?

Способ 2

Ввести тики прямо в семантику языка

- Преимущество: можно вообще избежать подпрограмм, неэквивалентных исходным (не нужны будут проверки условий корректности)
- Недостаток: стандартное равенство будет слишком сильным, нужно будет вводить более слабые отношения вроде эквивалентности с точностью до тиков

Способ 3

Усилить семантику преобразований: эквивалентные преобразования компактного представления должны сохранять свойство семантической эквивалентности всех подпрограмм
Например, при добавлении равенств можно убеждаться, что это не создаст пустых циклов